Directory Based Cache Coherence Verification Logic in CMPs Cache System

Mamata Dalui
Department of CSE
National Institute of Technology
Durgapur, WB, India 713209
mamata.06@gmail.com

Keshav Gupta
Department of CSE
National Institute of Technology
Durgapur, WB, India 713209
guptakeshav24@gmail.com

Biplab K Sikdar
Department of CST
Bengal Engineering and Science University
Shibpur, WB, India 711103
biplab@cs.becs.ac.in

ABSTRACT

This work reports a high speed protocol verification logic for Chip Multiprocessors (CMPs) realizing directory based cache coherence system. A special class of cellular automata (CA) referred to as single length cycle 2-attractor CA (TACA), has been introduced to identify the inconsistencies in cache line states of processors private caches. The introduction of CA segmentation logic ensures a better efficiency in the design by reducing the number of computation steps of the verification logic by a factor of the number of segments. The cache coherence verification for a system with limited directory has also been addressed. The TACA keeps trace of the coherence status of the CMPs' cache system and memorizes any inconsistent recording done during the processors' reference. Theory has been developed to realize quick decision on the cache coherency.

General Terms

Verification, Design, Theory

Keywords

Cache coherence, Chip Multiprocessors, Cellular automata, Directory based protocol, Fault detection, TACA

1. INTRODUCTION

The increasing number of cores in Chip Multiprocessors (CMPs), puts threats to the reliability and dependability of the design [9]. Efficient solution is, therefore, demanded to overcome the non-compliance of the existing solutions designed for the single processor chip. A number of works addressed the issue from different perspectives [11, 12].

In CMPs, maintaining coherency of shared data is of utmost necessity. The coherence is to ensure correctness of computation as well as is to maintain power efficiency of a system. The schemes ensuring coherency in CMPs with thousands of cores, through frequent communication along the global wires, are reported in [11, 12]. This communication among the L1 caches seriously affects the system performance as well as the energy usage [12].

In [12], a verification logic has been proposed to dynamically detect errors in the coherence controller (CC) logic. A cache coherence protocol Dico-CMP, a directory based cache coherence protocol, for tiled CMPs architecture, has been proposed in [10]. It avoids indirection by providing a block from the owner node and reduces the network traffic compared to broadcast-based protocols.

The above concerns, motivate us to develop a scheme to determine the accuracy in maintaining the data consistency of the CMPs cache system. The design should function at speed as well as is to be cost effective. To explore such a design, we consider cellular automata tool.

Cellular automata (CA) is is effectively employed in VLSI design as well as in testing domain [6]. In this work, we are in the advent to employ CA for the synthesis of efficient protocol verification logic for CMPs cache system. A special class of CA referred to as single length cycle 2-attractor cellular automata (TACA) has been introduced to identify the inconsistencies in cache line states of processors' private caches. The TACA analyzes the status of cache updates and settles to an attractor indicating any faulty recording of cache block/line status. The hardware realization of the CA based design enables quick decision on the cache coherency. The CA segmentation logic assures a better efficiency in the design by reducing the computation steps.

2. CA PRELIMINARIES

A cellular automaton (CA) consists of a number of cells organized in the form of lattice. It can be viewed as an autonomous finite state machine (FSM). In a two state 3-neighbourhood CA, each cell stores either 0 or 1 that refers to the present state (PS) S_i^t at time t of the cell i. The next state (NS) of the cell i at $(t+1)$ is $S_i^{t+1} = f_i(S_{i-1}^t, S_i^t, S_{i+1}^t)$, where S_{i-1}^t and S_{i+1}^t are the present states of the left neighbor and right neighbor of the i^{th} cell at time t and f_i is the next state function (Fig.1). The states of all the cells $\mathcal{S}^t = (S_1^t, S_2^t, \cdots, S_n^t)$ at t is the present state of the CA. Therefore, the next state of an $n-$cell CA is $S^{t+1} = (f_1(S_0^t, S_1^t, S_2^t), f_2(S_1^t, S_2^t, S_3^t), \cdots, f_n(S_{n-1}^t, S_n^t, S_{n+1}^t))$

The next state function f_i of the i^{th} CA cell can be expressed in the form of a truth table (Table 1). The decimal equivalent of the 8 outputs (NS) is called 'Rule' R_i. There are $2^8 = 256$ rules in 2-state 3-neighborhood CA. Two such rules 254 and 255 are illustrated in Table 1. The first row lists the possible 2^3 (8) combinations of present

MES '13, June 23 - 24 2013, Tel-Aviv, Israel

Table 1: Next state functions

PS	111	110	101	100	011	010	001	000	*Rule*
RMT	(7)	(6)	(5)	(4)	(3)	(2)	(1)	(0)	
NS	1	1	1	1	1	1	1	0	254
NS	1	1	1	1	1	1	1	1	255

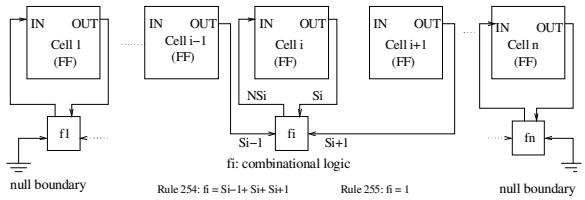

Figure 1: An n-cell null boundary CA

states S_{i-1}^t, S_i^t and S_{i+1}^t. The last two rows indicate the next states of the i^{th} cell at (t + 1), defining the rules 254 ($f_i = S_{i-1} + S_i + S_{i+1}$) and 255 ($f_i = 1$) respectively.

The rule vector R = $< R_1, R_2, \cdots, R_i, \cdots, R_n >$ configures the cells of a CA. If all the R_is are same -that is, R_1=R_2= $\cdots$ =R_n, the CA is a uniform CA; otherwise it is a non-uniform/hybrid CA. In Fig.1, the left (right) neighbor of the leftmost (rightmost) terminal cell is permanently fixed to 0-state. It is a null boundary CA. A CA is reversible if its states form only cycles in the state transition diagram (all states are reachable); otherwise, it is irreversible (Fig.2).

A combination of present states shown in the 1^{st} row of Table 1 is the Min Term of a 3-variable $S_{i-1}^t, S_i^t, S_{i+1}^t$ switching function and referred to as the Rule Min Term (RMT). The column 011 is the 3^{rd} RMT. The next states corresponding to this RMT are 1 for both the rules 254 and 255.

A set of states of a CA forms loop (cycle) (7→7 and 9→1→9 of Fig.2), called attractor. An attractor forms a basin with the states that lead to the attractor (7-basin of Fig.2 contains 12 states including the attractor state 7). The depth of a CA is defined as the length of the longest path from a state to an attractor in the state transition diagram. The depth of the CA shown in Fig.2 is 5 (2→10→6→4→5→7).

Definition 1. An RMT x0y (x1y) is called self-replicating if RMT x0y (x1y) is 0 (1). For example, RMT 0(000) is 0

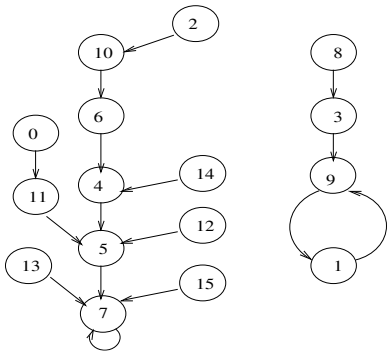

Figure 2: A 4-cell irreversible CA $< 1, 236, 165, 69 >$

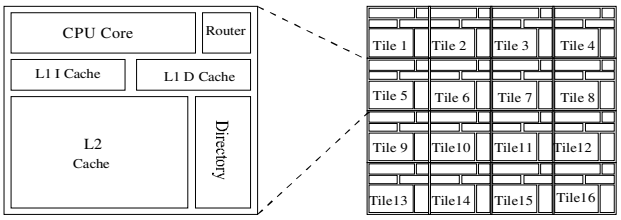

Figure 3: Tile CMPs organization

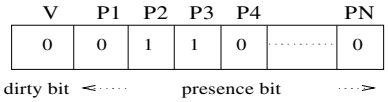

Figure 4: Sharing vector

and RMT 2(010) is 1 in 254 and, therefore, self-replicating.

3. CACHE COHERENCE IN CMPS

In Chip Multiprocessors (CMPs), the shared bus is replaced due to its large area requirement and bus contention. In this work, we consider the ATAC processor architecture [8] which uses a tiled multicore architecture (Fig.3) integrating on-chip optical network. It is a low-latency, energy-efficient, global and long distance communication network.

Each core in ATAC consists of private L1 and shared L2 caches. The L2 is typically NUCA. The increasing number of cores compels the designers to introduce larger on-chip caches for avoiding bottleneck at the off-chip interconnect. The last level cache incurs a considerable access latency depending on the distance from the chip. Sharing of data is, therefore, realized to reduce replication at the last level cache. It demands the shared data must be kept coherent.

The L1 cache miss in ATAC generates coherence messages. The other L1 caches of the system update their cache line states in accordance with the coherence messages. The cache coherence controller (CC) of the CMPs is entrusted to ensure the consistencies of shared data among the processor cores.

The snoop based cache coherence protocol is easy to implement but not so scalable. The bandwidth requirement in a snoop based system is also very high. It incurs a huge communication overhead. This in turn contributes to the total system power consumption. For large scale CMPs, the directory based cache coherence system is, therefore, desirable [10]. It issues coherence commands only to the caches that might be affected by an update.

A directory (Fig.4) consists of sharing bit vectors. Each vector maintains pointers to the processors that have copies of the data block in their caches. The directory also stores the state of cached copy. The 'd' specifies the dirty bit. d=1 (true) implies that there is a cache with the latest copy of B and the main memory copy is an invalid copy. The P_is are the presence bit. P_i=1 implies that the processor P_i is having the block B in its cache.

4. OVERVIEW OF PROPOSED DESIGN

The current design considers ATAC processor architecture following directory as well as snoopy protocols. The ATAC employs ACKwise [8], derived from MOESI protocol.

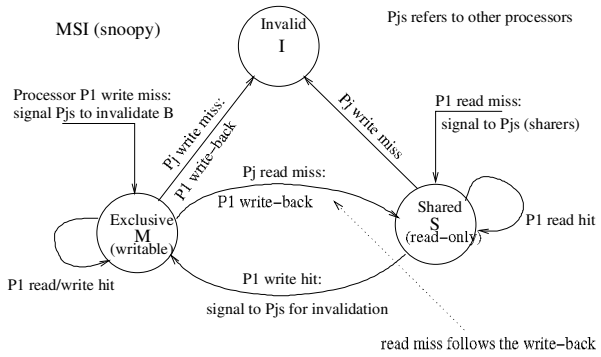

Figure 5: State transition diagram of MSI protocol

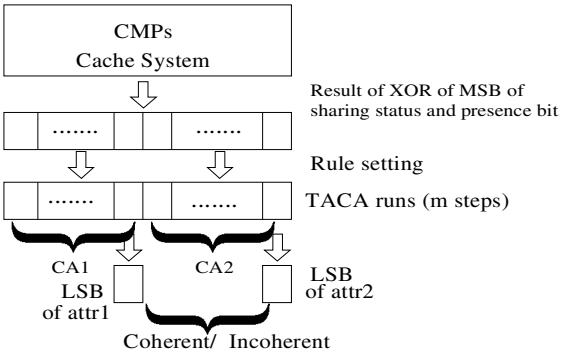

Figure 6: Segmented CA based verification unit

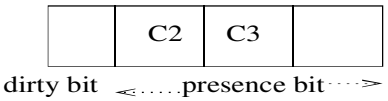

Figure 7: Sharing vector for limited directory

The cache coherence in CMPs relies on the performance of coherence controller (CC). A defect in CC can lead to faulty next state computation. There can also be the faults in communication network of the system. This issue is addressed in [12]. The solution dynamically detects errors in a coherence controller (CC). It involves complex data structures & computation intensive steps and considers only the snoop based protocol. In [2], [3], [4], we also report design of verification unit for the CC in snoopy protocol.

In a system with directory based protocols (e.g. ATAC), the updation of sharing bit vectors are to be considered. The sharing vector may also be subjected to faults, even if the sharing status of the cache blocks are recorded correctly.

Fig.5 describes the basic 3-state MSI protocol with full-map directory (Section 3). Table 2 [5] displays the effect of faulty noting in sharing vector, resulting in Faulty (F) state (the last column). The entry 'All Cs I[S]' represents that the cache line B at all the caches is in Invalid [Shared] state.

On a read/write operation (event) of a processor P_i, the status of sharing vector for B needs to be updated. The desired updated sharing vector for an event is shown in column 3 of Table 2. The possible faulty noting of the vector is represented in column 4. For example, in the row 1 of Table 2, column 1 notes the cache states prior to the event. The block B is in 'Invalid' state in all the caches. On an event of read by P_i (noted in column 2), the desired sharing vector is 'C_i S and all others I' (column 3). All the possible faulty recording of sharing vector are shown in column 4. Column 5 notes the effect of fault as 'Faulty' (F). The consideration of the columns 1 (cache states prior to the event), 4 and 5 of Table 2 indicates that the proposed fault detection unit should respond as F for the states of cache line B when -
Case 1: P_i reads/writes B, but P_i's presence bit in sharing vector for B is not updated to '1' - that is, it is still '0'.
Case 2: P_i reads/writes, instead of P_i's presence bit, P_j's presence bit in the sharing vector is set to '1'.
Case 3: P_i writes, some other processors' presence bits are not modified - that is, still '1'.

The CA-based logic, to realize the fault detection, is so designed that it can correctly respond either NF (non-faulty) or F (faulty) following the above three cases. It employs an n-cell CA [5] for a CMPs with n private (L1) caches (C1, C2, ...Cn, where Ci is the cache attached to processor Pi).

In CA segmentation, at each transition from a current cache state to the next state of Table 2 and corresponding updation of the presence bit in the sharing vector, the fault detection unit forms 2^p CA components each of $m = \frac{n}{2^p}$ cells. The rule R_i of the i^{th} CA cell corresponds to the compatibility of status of cache C_i with the i^{th} presence bit. For p=1, the compatibility status from C_1, C_2 ...$C_{n/2}$ forms CA1 and that of $C_{n/2+1}$...C_n forms CA2. The CA1 and CA2 are then run parallelly for $m = \frac{n}{2}$ time steps. The LSBs of CA1 & CA2 dictate an inconsistency if exists (Fig. 6). It effectively reduces the number of computation steps of the verification logic by a factor of the number of segments.

In a system realizing limited directory, the proposed solution can also effectively be employed for ensuring cache coherence. The structure of limited directory, shown in Fig. 7, indicates that the cache C2 and C3 (corresponding to processor P2 and P3) are currently sharing a block and at most three processors can share a particular block simultaneously. That is, the updation of sharing status and sharing bit vector is confined within a small region. Hence, the CA-based verification unit for limited directory can be realized only with an r-cell CA, where r is the number of sharers and $r << n$, and thereby reduces the time to decide on the coherence status.

5. TEST LOGIC DESIGN

An effective solution for the scheme described in the earlier section demands - the CA constructed should form single length cycle attractors, preferably two attractors (TACA), one corresponds to 'NF' and the other corresponds to 'F'. Further, the TACA should correctly report all the faulty cases of Section 4 (Table 2).

5.1 Single length cycle attractor

The next state of a single length cycle attractor is the attractor itself. In a single length cycle attractor CA, for at least one RMT (Section 2) of each cell rule $\mathcal{R}_i$ of R the cell i does not change its state in the next time step. It implies that the state change in cell i is $d \rightarrow d$. That is, for $\mathcal{R}_i$, if the RMT 0(000), 1(001), 4(100), 5(101) are 0, then the CA cell i, configured with $\mathcal{R}_i$, does not change its state. Similarly,

Table 2: Cache State Transitions

Current Cache States (1)	Event (2)	Desired Sharing Vector (3)	Faulty Sharing Vector (4)	Effect of Fault (5)	Cases (6)
All Cs I	Pi reads	Ci S and all others I	All Cs I	Faulty (F)	Case 1
			Cj S and all others I	Faulty (F)	Case 2
	Pi writes	Ci M and all others I	All Cs I	Faulty (F)	Case 1
			Cj M and all others I	Faulty (F)	Case 2
All Cs S	Pi writes	Ci M and all others I	Cj M and all others I	Faulty (F)	Case 2
			Ci & Cj M and others I	Faulty (F)	Case 3
			Ci M and others S & I	Faulty (F)	Case 3
Cs are I & S	Pi reads	Ci S and all others S & I	Ci I and others S & I	Faulty (F)	Case 1
	Pi writes	Ci M and all others I	Ci M and others S & I	Faulty (F)	Case 3
Cj M and all others I	Pi writes	Ci M and all others I	Ci & Cj M and others I	Faulty (F)	Case 3

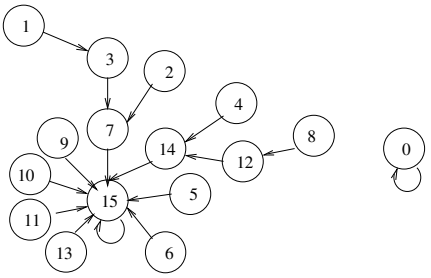

Figure 8: State transition of $< 254, 254, 254, 254 >$

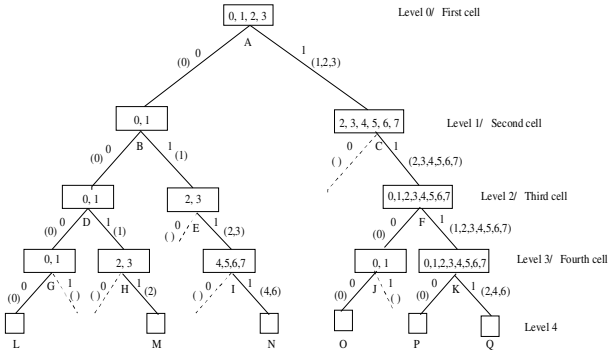

Figure 9: RT of the CA $< 254, 254, 254, 254 >$

if the RMTs 2(010), 3(011), 6(110) or 7(111) are 1 in a rule $\mathcal{R}_i$, a cell configured with $\mathcal{R}_i$ can stick to its current state in the next time step.

Property 1: [7] A rule $\mathcal{R}_i$ can lead to the formation of single length cycle attractor(s) if at least one of the RMTs 0, 1, 4 or 5 is 0, and/or at least one of the RMTs 2, 3, 6 or 7 is 1.

Based on Property1, the 256 CA rules are classified in 9 groups (group 0-8) in [7]. The rule 254 (11111110) is in group 5 as it follows Property1 for 5 RMTs. A CA configured with the rules that maintain Property1 for its RMTs is a probable candidate for single length cycle attractor CA. The construction of 2-attractor CA (Fig.8) can be followed from the theory of Reachability tree introduced in [7].

5.2 Reachability tree

Reachability Tree (RT) is a binary tree representing reachable states of a CA [7]. Each node is constructed with RMT(s) of a rule. The left edge is the 0-edge and the right edge is 1-edge (Fig.9). For an n-cell CA, the number of levels in the tree is (n+1). Root node is at Level 0 and the leaf/terminal nodes are at Level n. The nodes at Level i are constructed from the RMTs of $(i+1)^{th}$ CA cell rule $\mathcal{R}_{i+1}$. The decimal numbers within a node at level i represent the RMTs of the CA cell rule $\mathcal{R}_{i+1}$ based on which the cell (i+1) can change its state. The RMTs of a rule for which we follow 0-edge or 1-edge are noted in the bracket. The number of leaf nodes in an RT denotes the number of reachable states of the CA and a sequence of edges from the root to a leaf node, representing an n-bit binary string, is the reachable state. The 0-edge and 1-edge represent 0 and 1 respectively.

The RMTs of two consecutive cell rules $\mathcal{R}_i$ and $\mathcal{R}_{i+1}$ are related while the CA changes its state. This relationship

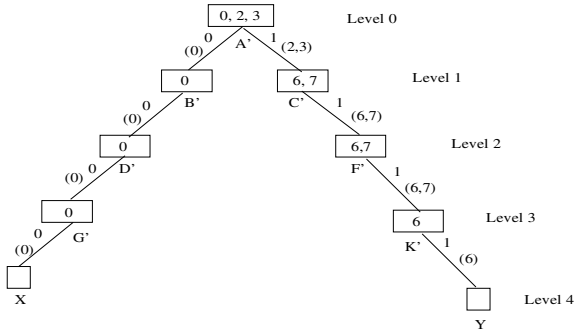

Figure 10: RT for attractors of $< 254, 254, 254, 254 >$

between the RMT of $\mathcal{R}_i$ and of $\mathcal{R}_{i+1}$ is shown in Table 3. It implies if the i^{th} CA cell changes its state following RMT K_i, then $(i+1)^{th}$ cell changes its state following RMT K_{i+1}.

Fig.9 is the reachability tree of a CA $<$ 254, 254, 254, 254 $>$. The root node (level 0) of Fig.9 is constructed from RMTs 0, 1, 2 and 3 as cell 1 (rule 11111110) can change its state following any one of the RMTs 0, 1, 2 and 3. As the state of left neighbor of cell 1 is always 0, the RMTs 4, 5, 6 & 7 are the don't cares for cell 1. Similarly, as the right neighbour of cell 4 is always 0, the RMTs 1, 3, 5 & 7 are don't care for cell 4. It is obvious from Fig.9 that there are 6 possible sequences of edges from root to leaf indicating 6, out of 16, CA states are reachable and rests are non-reachable.

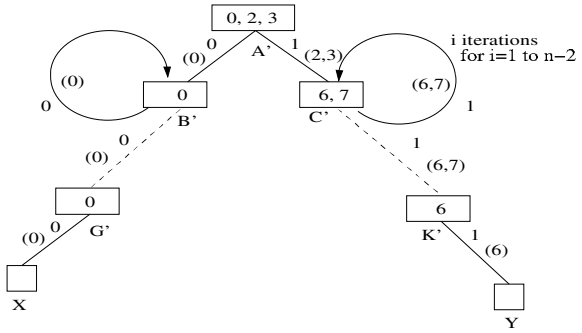

Figure 11: RT for attractors of $< 254, 254, ..., 254 >$

An RT (Fig.9) can also be modified to display only the single length cycle attractors. Since all the RMTs of a cell rule can not contribute to generate attractors, such (insignificant) RMTs can be removed from the nodes of RT [7]. For example, RMT 2 (010) is 0 implies that it is insignificant. The tree shown in Fig.10 corresponds to the CA $<$ 254, 254, 254, 254 $>$. It is derived from Fig.9 to point to the single length cycle attractors (X and Y) only.

RT for CA of arbitrary length: The RT for attractors in Fig.10 can be generalized to depict the attractors for an n-cell CA. The generalized RT for attractors of the CA $<$ 254, 254, ..., 254$>$ is shown in Fig.11. It can be observed that the 0-edge at B' of Fig.10 evolves to the node D' with same set of RMTs ({0}) that is node B' and D' are equivalent and, therefore, transition B' to D' is replaced by the transition B' to B' in Fig.11. Similarly, the transition C'$\rightarrow$ F' in Fig.10 is replaced by the transition C' $\rightarrow$ C' in Fig.11. Such transitions between equivalent states are true for level 1 to level (n-2). For the last cell of a CA (<254, 254, ..., 254>) some of the RMTs of B' and C' (RMT 7 of C') are don't cares. Therefore, level (n-1) is shown separately (K' in Fig.11). The RT for attractors of Fig.11 depicts that the n-cell uniform CA with rule 254 forms 2-single length cycle attractors (all 0s and all 1s).

5.3 Selection of CA for the design

This subsection reports the properties of TACA rules that are found to be essential for our TACA based design.

Theorem 1. The rules of group 0, 1, 2, 6, 7, and 8 can't form TACA.

Proof. The proof is in [1]. □

Theorem 2. In 3-neighbourhood null-boundary, the uniform CA constructed with only 16 rules (out of 256) can generate all 0s and all 1s single length cycle attractors.

Proof. The proof is in [1]. □

Theorem 1 & 2 define that there are only 5 rules 218, 234, 248, 250 & 254 that may form TACA with all 0s and all 1s single length cycle attractors [1]. The outcome of extensive experimentation leads to following observations.

Observation 1. In group 3, the uniform CA designed only with the rule 38 and 52 can be the TACA.

Observation 2. In group 4, the rules 46, 106, 116, 120, 166, 180, 235 and 249 only form uniform TACA.

Table 3: Relationship between RMTs of cell i and cell (i+1) for next state computation

RMT K_i at i^{th} rule	RMTs K_{i+1} at $(i+1)^{th}$ rule
0/4	0, 1
1/5	2, 3
2/6	4, 5
3/7	6, 7

Observation 3. The uniform CA designed only with the rules 174, 239, 244, 253, and 254 of group 5 be the TACA.

The above observations lead to the following theorem.

Theorem 3. The uniform CA with rule 254 has only two single length cycle attractors (all 0s and all 1s states).

Proof. The proof is in [1]. □

5.4 The test logic

Let, the attractors of CA formed for all the cases of column 3 (NF cases) of Table 2 are X (all 0s) and for other cases (F cases) the CA formed has the only attractor Y (all 1s). The best selection of rules, therefore, should be such that

Cond 1: $A_1 = \{X\}$ belongs to NF and

$A_2 = \{Y\}$ belongs to F are different.

For the current design, we consider a uniform CA with rule 254. In the NF case, the CA representing the cache system, traces through the X-attractor basin. Now, if there is a fault, the CA traces through the Y-basin. Since the CA configured with 254 is a TACA (Fig.8), for the NF case, the CA settles to an attractor (00...0) with LSB of attractor as '0'. In the F case, it settles to attractor Y with LSB as '1'. Hence, LSB of attractor '0' signifies "Non-faulty" recording and '1' signifies a "Faulty" recording.

In the sharing status vector the states M, S and I of a cache line are represented as the 11, 10, and 00 respectively. For a cache block B, if B is in M or in S state in the cache C_i, then the i^{th} bit of the sharing vector is 1. That is, in a nonfaulty case, the MSB of sharing status for C_i and the i^{th} bit of the sharing vector are same, either 1 or 0. The CA accepts XOR of the MSB of 2-bit sharing status (0/1) of B at C_i and the i^{th} bit of sharing vector for B as the initial state of the CA cell i. That is, the outcome of XOR operation is used as the initial seed for an n-cell uniform CA configured with rule 254. The CA is then run for $t(=n)$ time steps and LSB of the attractor dictates the presence of fault(s) in either sharing status or in sharing vector.

The hardware realization of the design is shown in Figure 12. If the cache line B's state at i^{th} cache C_i is 'Invalid' -that is, $P_{i1}P_{i0}$=00 and the i^{th} bit in sharing vector is '0', then $0 \oplus 0$ = '0', thereby, reaching the attractor $<$ 00...0 $>$ of the uniform CA. However, with B's state at i^{th} cache C_i as 'Invalid' -that is, $P_{i1}P_{i0}$=00 and the i^{th} bit in sharing vector is '1', then the $0 \oplus 1$ is '1', thus, reaching the attractor $<$ 11...1 $>$. Hence, the LSB of the attractor '0' indicates the absence of fault and LSB '1' indicates presence of single or multiple faults at different positions.

6. HIGH SPEED VERIFICATION

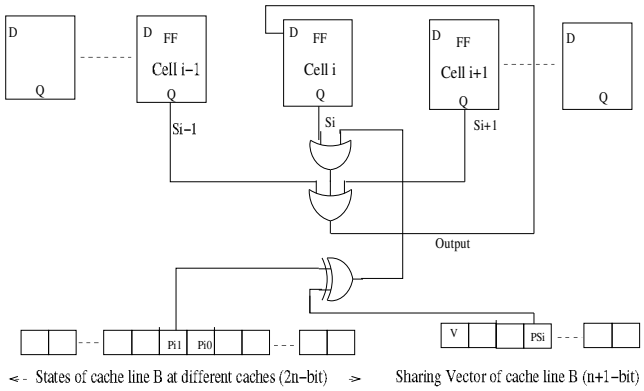

Figure 12: Realization of fault detection unit

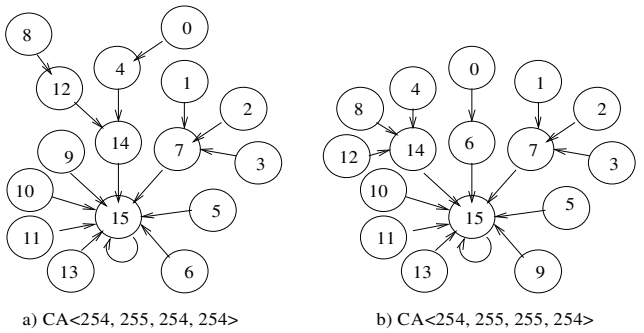

Figure 13: State transition after hybridization

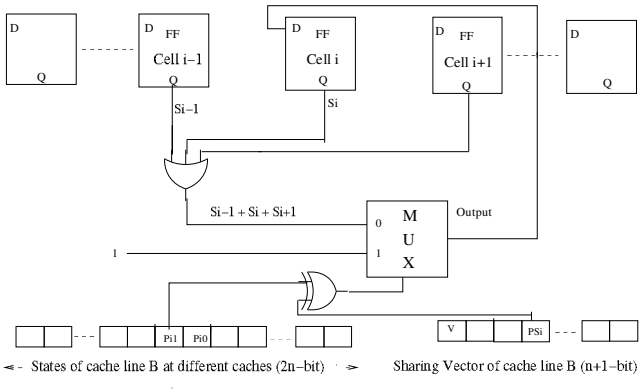

Figure 14: Realization of high speed fault detection

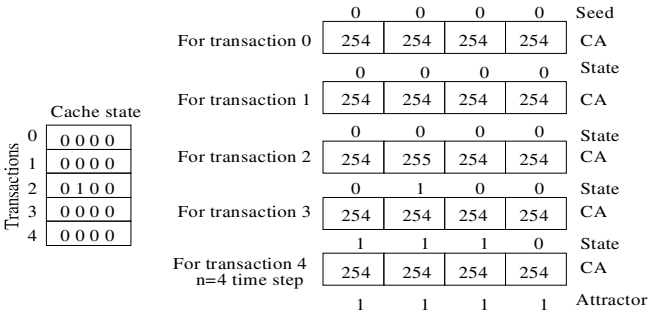

Figure 15: Functioning of the test hardware

The scheme in the earlier section reports faults on each transaction. In this section, we propose a design that can check whether one or more of the transactions within a set of transactions result in a fault.

On each transaction, MSB of sharing status for ith cache C_i is XORed with ith presence bit of the sharing bit vector. The result of XOR operation sets i^{th} CA cell rule of the n-cell CA chosen for the design. The rule R_o is set for '0' and rule R_h for a '1'. Then the CA is run for 1 time step. The next state NS_j of the CA configured for j^{th} transaction is then used as the seed for the CA configured for the $(j+1)^{th}$ transaction. If the status of verification is needed to be observed after k transactions, then after such k steps the k^{th} CA formed is run for (t=) n-step. The LSB ('0'/'1') of the attractor reached indicates whether the sequence of k transactions has passed through all safe states.

Theorem 4. Uniform TACA configured with rule 254 (R_o) is converted into an SACA when hybridized with rule 255 (R_h) at single or multiple positions.

Proof. Uniform TACA configured with rule 254 forms two attractors, X=00...0 and Y=11...1 (Theorem 3) i.e. the RT for attractors has only two branches leading to X & Y. The self-replicating RMTs of rule 254 are RMT 0, 2, 3, 6 & 7 and that of rule 255 are 2, 3, 6 & 7. Now, if hybridized with rule 255 at any position of the uniform CA <254, 254, ..., 254>, it would block the '0' branch (all 0s attractor) since the RMT 0 for rule 255 is not self-replicating. □

Theorem 5. The TACA rules for which RMT 0 and RMT 7 are self-replicating and one of RMT 1 or 3 and one of RMT 4 or 6 are self replicating, can be converted to an SACA when hybridized with rule 255 at single or multiple positions.

Proof. The proof is in [1]. □

For the current realization, we consider hybridization (with R_h = 255) of a uniform CA with rule R_o = 254. The hybridization allows merger of attractor basins of the uniform CA. That is, for NF case of Section 4, the system traces through the X-basin (0-basin, Fig.8) of the uniform CA and for a fault, the hybrid CA is formed. The system traces through the merged Y-basin (15-basin, Fig.13) of hybrid CA.

The hardware realization of the design is shown in Fig.14 (the states M, S and I of a cache line are represented by 11, 10 and 00). The 2 to 1 multiplexer (MUX) and the other combinational logic (shown in the figure) generate the next state of a CA cell. For example, consider a set with k number of transactions in a cache line. Say, for the very first transaction, the cache line B's state at i^{th} cache C_i is 'Invalid' -that is, $P_{i1}P_{i0}$=00 and the i^{th} bit in sharing vector is '0', then 0⊕0 = '0' selects rule for the cell i as 254. The CA runs for 1-step with all 0s initial seed. The next state NS_1 is used as seed for the CA configured for the second transaction. Say, for ith transaction, the cache line C's state at j^{th} cache C_j is 'Invalid' -that is, $P_{j1}P_{j0}$=00 and the j^{th} bit in sharing vector is '1', then 0⊕1 = '1'. It selects rule for the cell j as 255. In Fig.14, this sets the select lines of the MUX as 1 -that is, rule 255. Finally, for the kth transaction, the CA is configured with appropriate rules and is run for n-step. The LSB of attractor '1' indicates the occurrence of one or more unsafe states. The attractor's LSB '0' indicates the set of k transactions is passed through safe states.

Table 5: Performance in CA segmentation

No. of Proc	Scheme [5]		Proposed scheme		
	No. of comp. step	No. of check bits	No. of segments	No. of comp. step	No. of check bits
64	64	1	2	32	2
			4	8	4
256	256	1	2	128	2
			4	64	4
1024	1024	1	2	512	2
			4	256	4
			8	128	8

Fig.15 describes the operation of the test logic. Let us consider a system of 4 caches and a set of 5 transactions on the caches among which the 2^{nd} transaction results in a fault. With the 0^{th} transaction, a 4-cell uniform CA is formed with rule 254. It is run for 1-time step with all 0s seed. Its next state is an all 0s state. Since transaction 1 is also nonfaulty, the CA formed for this is also the uniform CA with rule 254 and the next state produced is all 0s state. Transaction 2 is a faulty transaction, resulting in a hybrid CA configured with rule 254 and 255. Its next state is 0100. The CA constructed for transaction 3 and 4 are also the uniform CA. The CA for transaction 3 results in 1110 state. The CA for transaction 4 is then run for 4 step and it reaches the attractor 1111 state.

7. EXPERIMENTAL RESULTS

This section reports the experimental results establishing the effectiveness of the CA based test design reported in this work. In the experimentation, we consider CMPs with 16 to 1024 number of processors (column 1 of Table 4). For each such system, we generate possible sequence of states of a cache line B at different private (L1) caches. Column 2 of Table 4 notes the chosen sequence of cache states. Column 3 notes the event that might cause state transition of the cache states. For an n (number of processors), we choose all the possible cases (Case 1- Case 3 of section 4) with a read or write operation. The correct recording of sharing status has been noted in column 4 and column 5 notes possible faulty recording of sharing vectors. The MSB of sharing status for each cache is XORed with the corresponding presence bit in the sharing vector and the result of this operation is shown in column 6. The CA rules configured for the CA, are shown in column 7. The attractors are noted in column 8. The LSB of the attractors' ('0'/'1') are indicated by boldfaces. It can be observed that the LSB of the attractor correctly indicates whether the cache states are correctly recorded in the sharing bit vector ('0' for non-faulty(NF), '1' for faulty(F)). The decision is noted in the last column.

Further, Table 5 reports the performance improvement achieved through introduction of CA segmentation logic. For the experimentation, we have considered 64 to 1024 number of processors. The column 1 of Table 5 reports the number of processors, the column 2 reports the number of computation steps in [5]. Column 3 shows the number of check bits to decide on the incoherency. The columns 4-6 report the number of segments, computation steps and the check bits required in the proposed scheme.

8. CONCLUSION

This work reports an efficient solution for quick determination of data inconsistencies in caches as well as in sharing vectors of a system realizing the directory based cache coherence protocol. It avoids rigorous computational overhead assuring robust and scalable design specially in a system with thousands of processor cores. The proposed design is developed around a the regular structure of cellular automata (CA). The segmentation of CA is done to ensure better efficiency in both the systems with full directory as well as limited directory protocol.

9. REFERENCES

[1] M. Dalui. Theory and applications of cellular automata for cmps cache system protocol design and test. In PhD Thesis Proposal submitted). BESU, West Bengal, India, July 2012.

[2] M. Dalui and B. K. Sikdar. An efficient test design for verification of cache coherence in cmps. In 9th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2011), December 2011.

[3] M. Dalui and B. K. Sikdar. An efficient test design for cmps cache coherence realizing mesi protocol. In VDAT, July 2012.

[4] M. Dalui and B. K. Sikdar. A test design for quick determination of incoherency in chip multiprocessors' cache realizing moesi protocol. In ISED, Dec 2012.

[5] M. Dalui and B. K. Sikdar. Design of directory based cache coherence protocol verification logic in cmps around taca. In International Conference on High Performance Computing and Simulation (Accepted), July 2013.

[6] S. Das, D. Dey, S. Sen, B. K. Sikdar, and P. P. Chaudhuri. An efficient design of non-linear ca based prpg for vlsi circuit testing. In In Proceedings of ASP-DAC, pages 110–112, January 2004.

[7] S. Das, S. Mukherjee, N. N. Naskar, and B. K. Sikdar. Characterization of single cycle ca and its application in pattern classification. Journal of Electronic Notes in Theoretical Computer Science (ENTCS), 252:181–203, October 2009.

[8] G. Kurian, J. E. Miller, J. Psota, J. Eastep, and J. Liu. Atac: A 1000-core cache-coherent processor with on-chip optical network. In Parallel Architectures and Compilation Techniques (PACT), Sept 2010.

[9] K. Olukotun and L. Hammond. The future of microprocessor. Magazine Queue - Multiprocessors., 3(7):26–29, September 2005.

[10] A. Ros, M. E. Acacio, and J. M. Garca. A direct coherence protocol for many-core chip multiprocessors. IEEE Trans on Parallel and Distributed Ststems, 21(12):341–348, December 2010.

[11] P. Subramanyan, V. Singh, K. K. Saluja, and E. Larsson. Energy-efficient fault tolerance in chip multiprocessors using critical value forwarding. In 201O IEEE/IIFIP International Conference on Dependable Systems & Networks (DSN), pages 121–130, June 2010.

[12] H. Wang, S. Baldawa, and R. Sangireddy. Dynamic error detection for dependable cache coherency in multicore architecture. In VLSI Design conference, January 2008.

Table 4: Experimental Results

No. of proc (1)	Cache states (2)	Event (3)	Sharing status(SS) (4)	Sharing Vector SV) (5)	MSB of SS $\oplus$ SV (6)	CA Rules (7)	Attractor (8)	Remark (9)
16	All Cs are I	P_2 read	00 10..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 10..00 00	0 0..1 0	0 0..1 0	254 254...254	1..1	F
		P_2 write	00 11..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 11..00 00	0 0..1 0	0 0..1 0	254 254...254	1..1	F
			00 11..00 00	0 1..1 0	0 1..1 0	254 254...254	1..1	F
	All Cs are S	P_2 write	00 11..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 11..00 00	0 0..1 0	0 1..1 0	254 254...254	1..1	F
			00 11..00 00	0 1..1 0	0 0..1 0	254 254...254	1..1	F
			00 11..00 00	0 1..0 1	0 0..0 1	254 254...254	1..1	F
	C_2 S & others I	P_2 read	00 10..00 10	0 1..0 1	0 0..0 0	254 254...254	0..0	NF
			00 10..00 10	0 0..0 1	0 1..0 0	254 254...254	1..1	F
		P_2 write	00 11..00 00	0 1..0 1	0 0..0 1	254 254...254	1..1	F
	C_0 M & others I	P_2 write	00 11..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 11..00 00	1 1..0 0	0 0..0 0	254 254...254	1..1	F
			00 11..00 00	1 0..0 0	0 0..0 0	254 254...254	1..1	F
256	All Cs are I	P_2 read	00 10..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 10..00 00	0 0..1 0	0 0..1 0	254 254...254	1..1	F
		P_2 write	00 11..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 11..00 00	0 0..1 0	0 0..1 0	254 254...254	1..1	F
			00 11..00 00	0 1..1 0	0 1..1 0	254 254...254	1..1	F
	All Cs are S	P_2 write	00 11..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 11..00 00	0 0..1 0	0 1..1 0	254 254...254	1..1	F
			00 11..00 00	0 1..1 0	0 0..1 0	254 254...254	1..1	F
			00 11..00 00	0 1..0 1	0 0..0 1	254 254...254	1..1	F
	C_2 S & others I	P_2 read	00 10..00 10	0 1..0 1	0 0..0 0	254 254...254	0..0	NF
			00 10..00 10	0 0..0 1	0 1..0 0	254 254...254	1..1	F
		P_2 write	00 11..00 00	0 1..0 1	0 0..0 1	254 254...254	1..1	F
	C_0 M & others I	P_2 write	00 11..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 11..00 00	1 1..0 0	0 0..0 0	254 254...254	1..1	F
			00 11..00 00	1 0..0 0	0 0..0 0	254 254...254	1..1	F
1024	All Cs are I	P_2 read	00 10..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 10..00 00	0 0..1 0	0 0..1 0	254 254...254	1..1	F
		P_2 write	00 11..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 11..00 00	0 0..1 0	0 0..1 0	254 254...254	1..1	F
			00 11..00 00	0 1..1 0	0 1..1 0	254 254...254	1..1	F
	All Cs are S	P_2 write	00 11..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 11..00 00	0 0..1 0	0 1..1 0	254 254...254	1..1	F
			00 11..00 00	0 1..1 0	0 0..1 0	254 254...254	1..1	F
			00 11..00 00	0 1..0 1	0 0..0 1	254 254...254	1..1	F
	C_2 S & others I	P_2 read	00 10..00 10	0 1..0 1	0 0..0 0	254 254...254	0..0	NF
			00 10..00 10	0 0..0 1	0 1..0 0	254 254...254	1..1	F
		P_2 write	00 11..00 00	0 1..0 1	0 0..0 1	254 254...254	1..1	F
	C_0 M & others I	P_2 write	00 11..00 00	0 1..0 0	0 0..0 0	254 254...254	0..0	NF
			00 11..00 00	1 1..0 0	0 0..0 0	254 254...254	1..1	F
			00 11..00 00	1 0..0 0	0 0..0 0	254 254...254	1..1	F